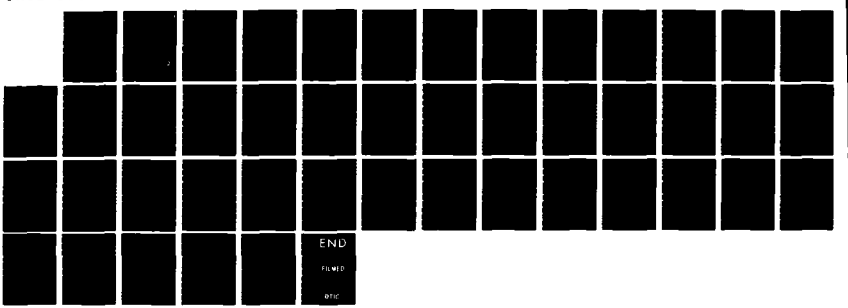
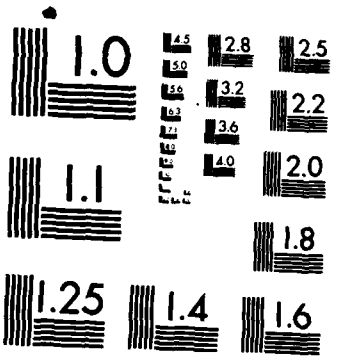
AD-A162 661 SIMULATIONS AMONG MULTIDIMENSIONAL ITERATIVE ARRAYS 1/1
ITERATIVE TREE AUTOMA (U) ILLINOIS UNIV AT URBANA
COORDINATED SCIENCE LAB J L TRAHAN JAN 86
UNCLASSIFIED UILU-ENG-86-2202 F/G 12/1 NL





January 1986

UILU-ENG-86-2202
ACT-66

(12)

COORDINATED SCIENCE LABORATORY
College of Engineering

SIMULATIONS AMONG MULTI-DIMENSIONAL ITERATIVE ARRAYS, ITERATIVE TREE AUTOMATA, AND ALTERNATING TURING MACHINES

Jerry Lee Trahan

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

ADA 162 661

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-86-2202 (ACT-66)	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Laboratory University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State and ZIP Code) 1101 W. Springfield Avenue Urbana, Illinois 61801	7b. ADDRESS (City, State and ZIP Code) 800 N. Quincy Street Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research 8b. OFFICE SYMBOL (If applicable) N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Contract # N00014-85-K-0570
8c. ADDRESS (City, State and ZIP Code) 800 N. Quincy Street Arlington, VA 22217	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: N/A PROJECT NO.: N/A TASK NO.: N/A WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) Simulations among multidimensional iterative arrays, iterative	
12. PERSONAL AUTHOR(S) Trahan, Jerry L.	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1986 January 15. PAGE COUNT 42
16. SUPPLEMENTARY NOTATION N/A	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) iterative array, alternting turing machine, parallel computation, simulation, computational complexity theory. (5 THESIS)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We present three simulations: a simulation of an alternating Turing machine (ATM) operating in time T(n) by an iterative tree automation (ITA) ~~in time O(T(n))~~, a simulation of a d-dimensional iterative array (dIA) operating in time T(n) by an ATM ~~in time~~ $O((T(n))^{d/2})$, and a simulation of an ITA operating in time T(n) by an ATM ~~in time~~ $O((T(n))^{2})$. The first two improve previously known results. The first implies the simulation of a nondeterministic Turing machine by an ITA in time O(T(n)) of Culik and Yu (1984). The second is stronger than the simulation of a dIA by an ATM in time $O((T(n))^{d+1}/\log T(n))$ of Seiferas (1977) and Dymond and Tompa (1985).

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) 22c. OFFICE SYMBOL NONE

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

11. tree automata, and alternating turing machines

SIMULATIONS AMONG MULTIDIMENSIONAL ITERATIVE ARRAYS. ITERATIVE TREE AUTOMATA. AND ALTERNATING TURING MACHINES

BY

JERRY LEE TRAHAN

B.S., Louisiana State University and A. & M. College, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1986

Urbana, Illinois

ABSTRACT

We present three simulations: a simulation of an alternating Turing machine (ATM) operating in time $T(n)$ by an iterative tree automaton (ITA) in time $O(T(n))$, a simulation of a d-dimensional iterative array (dIA) operating in time $T(n)$ by an ATM in time $O((T(n))^d)$, and a simulation of an ITA operating in time $T(n)$ by an ATM in time $O((T(n))^2)$. The first two improve previously known results. The first implies the simulation of a nondeterministic Turing machine by an ITA in time $O(T(n))$ of Culik and Yu (1984). The second is stronger than the simulation of a dIA by an ATM in time $O((T(n))^{d+1}/\log T(n))$ of Seiferas (1977) and Dymond and Tompa (1985).

ACKNOWLEDGEMENTS

I wish to thank my thesis advisor, Dr. Michael C. Loui, for his encouragement, guidance, and constructive suggestions for this thesis. I also wish to thank my parents for their constant support throughout my education. This work was supported in part by the Office of Naval Research under contract N00014-85-K-0570.

TABLE OF CONTENTS

LIST OF FIGURES

Page

Chapter 1

INTRODUCTION

Multidimensional iterative arrays, iterative tree automata, and alternating Turing machines are important models of parallel computation. A *d-dimensional iterative array* (dIA) comprises an infinite set of finite state machines located at the points of the d-dimensional integer lattice. An *iterative tree automaton* (ITA) consists of an infinite set of finite state machines connected into a binary tree. An *alternating Turing machine* (ATM), like a *nondeterministic Turing machine (NTM)*, may have choices of transitions for each combination of state, input symbol, and worktape symbols. From an *existential* state an ATM accepts if at least one choice leads to an accepting state; from a *universal* state an ATM accepts if every choice leads to an accepting state. One can view the ATM as a computational model that makes copies of itself to evaluate each of the choices. A *deterministic Turing machine (DTM)* has only a single possible transition for each combination of state, input symbol, and worktape symbols. Each of these models has a fixed structure. Each processor (finite state machine or ATM copy) can communicate with only a fixed set of other processors, in contrast to variable structure models such as the hardware modification machines of Dymond and Cook (1980) and the parallel random access machines of Fortune and Wyllie (1978).

This thesis studies the simulation of a dIA by an ITA. The simulation of an ATM by an ITA and the simulation of a dIA by an ATM achieve this purpose. Let X(t) denote the set of all languages recognized by machines of type X in time O(t), where X $\in$ {dIA, ITA, ATM, DTM, NTM}. These simulations produce the following time bounds:

$$\mathrm{ATM}(t) \subseteq \mathrm{ITA}(t) \text{ and}$$

$$\mathrm{dIA}(t) \subseteq \mathrm{ATM}(t^d).$$

In addition, this thesis presents the simulation of an ITA by an ATM within the bound

ITA(t) $\subseteq$ ATM (t^2), and considers the simulation of an ATM by a dIA.

Two of the above time bounds improve previously known results. The first, ATM(t) $\subseteq$ ITA(t), implies the bound NTM(t) $\subseteq$ ITA(t) established by Culik and Yu (1984) because an NTM is more restricted than an ATM. The second, dIA(t) $\subseteq$ ATM(t^d), is better than dIA(t) $\subseteq$ ATM($t^{d+1}/\log t$). The latter bound arises from the combination of a result of Seiferas (1977a), who proved that dIA(t) $\subseteq$ DTM(t^{d+1}), and a result of Dymond and Tompa (1985), who established that DTM(t) $\subseteq$ ATM(t/log t).

Aside from improving upon previous findings, the results described in this thesis are significant because they extend current knowledge about ATMs. In addition to the usual computational resources time and space (which generally are inversely related), the ATM has alternations between universal and existential states as a computational resource. The ATM simulation of the dIA and the ATM simulation of the ITA utilize the resource alternation to achieve the stated time bounds.

The ITA simulation of an ATM uses the ITA's capability to emulate direct central control, that is, to act as though the state transitions of each finite state machine in the ITA depend on the state of a uniquely designated finite state machine. The techniques of Seiferas (1977b) are used to establish the direct central control capability of the ITA.

The ATM simulation of the dIA uses the "divide-and-conquer" technique, as in Savitch (1970), Paterson (1972), and Loui (1981).

The framework of this thesis is as follows. Chapter 2 gives definitions of the dIA, the ITA, and the ATM. Chapter 3 reviews the literature related to the thesis research. Chapters 4 and 5 detail the simulations themselves, prove their correctness, and determine their running times. Chapter 6 describes the conclusions reached from this research and offers some open research problems.

Chapter 2

DEFINITIONS

To more precisely describe the actions of the computational models in the simulations to follow, this chapter defines the dIA, the ITA, and the ATM.

Let n be the length of the input string. For all n, a machine recognizes an input string of length n in time $T(n)$ if the machine requires at most T(n) steps to accept the input string.

A *d-dimensional iterative array* (dIA) is an infinite synchronized d-dimensional array of finite state machines, one for each d-tuple of nonnegative integers. In dIA M let M($\underline{X}$) denote the machine corresponding to the d-tuple $\underline{X}$, called the *coordinates* of M($\underline{X}$). Let $\underline{0}$ denote the d-tuple of all 0s. Only M($\underline{0}$) receives input symbols and produces output symbols. At each step M($\underline{0}$) reads a new symbol of the input string or does not read a new symbol as a function of the current state of itself, the current states of its neighbors, and the current input symbol. A *neighbor* of a machine with coordinates $\underline{X}$ in the array is a machine whose coordinates are obtained by adding or subtracting 1 from a single coordinate of $\underline{X}$. The state transitions of every machine depend on the current states of the machine itself and its 2d neighbors. The state transitions of M($\underline{0}$) depend additionally on the current input symbol. A dIA *accepts* an input string in Σ^* when M($\underline{0}$) enters a state in a designated set of final states.

Formally, a dIA M is a septuple $M = (d, Q, \Sigma, \delta_0, \delta, r, F)$ where

d is a positive integer,

Q is a finite set of states,

Σ is a finite input alphabet ($\$ \in \Sigma$ is an endmarker, $B \in \Sigma$ is the blank symbol),

δ_0: $Q^{2d+1} \times (\Sigma \cup \{B, \$\}) \rightarrow Q$ is the transition function for M($\underline{0}$), the machine at the origin,

δ: $Q^{2d+1} \rightarrow Q$ is the transition function for every machine other than M($\underline{0}$),

r: $Q^{2d+1} \times (\Sigma \cup \{B, \$\}) \rightarrow \{true, false\}$ is the function that specifies whether M($\underline{0}$) reads the

next input symbol, and

$F \subseteq Q$ is a set of final states.

$q_\lambda \in Q$ is a special *quiescent state*. Initially, every machine is in a quiescent state. δ satisfies the condition $\delta(q_\lambda,q_\lambda,\ldots,q_\lambda)=q_\lambda$ thus except for M(Q), a machine leaves the quiescent state only after a neighbor leaves the quiescent state. Infinite B's are assumed to follow the end of the input string.

An *iterative tree automaton* (ITA) is composed of synchronized finite state machines connected in an infinite full binary tree structure. Let $R(\lambda)$ denote the root of ITA R, where λ is the empty string. In general, for any finite binary string β let $R(\beta 0)$ and $R(\beta 1)$ denote the left and right children of $R(\beta)$, respectively. The *level* of $R(\beta)$ is $|\beta|$, the length of β. Define $|\lambda|=0$. Only $R(\lambda)$ receives input symbols and produces output symbols. At each step $R(\lambda)$ reads a new symbol of the input string or does not read a new symbol as a function of the current state of the root, the current states of its two children, and the current input symbol. The current state of the root, the current states of its two children, and the current input symbol determine the next state of $R(\lambda)$. For every other machine in the ITA, the current states of the machine itself, its parent and its left and right children determine the next state of the machine. An ITA *accepts* an input string when $R(\lambda)$ enters any one of a designated set of final states.

Formally, an ITA R is a septuple $R = (Q, \Sigma, \delta_0, \delta_1, \delta, r, F)$ where

Q is a finite set of states,

Σ is a finite input alphabet ($S \in \Sigma$ is an endmarker, $B \in \Sigma$ is the blank symbol),

$\delta_0: Q \times (\Sigma \cup \{B, S\}) \times Q^2 \rightarrow Q$ is the transition function for $R(\lambda)$, the machine at the root,

$\delta, \delta_1: Q^4 \rightarrow Q$ are the transition functions of the left and right children, respectively, of each machine $R(\beta)$, $\beta \in \{0,1\}^*$,

$r: Q \times (\Sigma \cup \{B, S\}) \times Q^2 \rightarrow \{true, false\}$ is the function that specifies whether the root reads the next input symbol, and

$F \subseteq Q$ is a set of final states.

If X, W, Y, Z are respectively the current states of $R(\beta 0)$, its parent $R(\beta)$, its left child $R(\beta 00)$, and its right child $R(\beta 01)$, then $\delta_l(X,W,Y,Z)$ is the next state of $R(\beta 0)$. Similarly δ_r specifies the next state of each $R(\beta 1)$.

$q_\lambda \in Q$ is a special *quiescent state*. Initially, every machine in the ITA is in a quiescent state. δ satisfies the condition $\delta(q_\lambda,q_\lambda,q_\lambda,q_\lambda)=q_\lambda$ where δ is δ_l or δ_r thus except for $R(\lambda)$, a machine leaves the quiescent state only after its parent leaves the quiescent state. Infinite B's are assumed to follow the input string. The transition function for a left child must be different from the transition function for a right child for the ITA to be distinct from a 1IA.

A *configuration* of an ITA R is a pair (C,w), where C is a mapping from the machines in R to Q and $w \in \Sigma^*\$$. For all $\beta \in \{0,1\}^*$, $X \in \{0,1\}$, where w is the unread portion of the input string, R has a *legal transition* from (C,w) to (C',w') if

(1) $C'(\lambda)=\delta_0(C(\lambda),a,C(0),C(1))$,

(2) $C'(\beta 0)=\delta_l(C(\beta 0),C(\beta 00),C(\beta 01),C(\beta))$,

(3) $C'(\beta 1)=\delta_r(C(\beta 1),C(\beta 10),C(\beta 11),C(\beta))$, and

(4) $w=w'=a=\$$, or

$w=aw'$ if $r(C(\lambda),C(0),C(1),a)=true$, or

$w=w'$ if $r(C(\lambda),C(0),C(1),a)=false$.

A *computation* by ITA R is a sequence (C_0,w_0), (C_1,w_1), (C_2,w_2), ... of configurations where the transition from (C_k,w_k) to (C_{k+1},w_{k+1}) is legal for all k. Note that $C_k(\beta)=C_0(\beta)$ for $|\beta| \geq k$ since we start with all machines in a quiescent state.

An *alternating Turing machine* (ATM) is defined in Chandra, Kozen, and Stockmeyer (1981). A *configuration* of an ATM is the state of the ATM, the contents of the input tape, the contents of each of the worktapes, and the locations of each of the tape heads. An ATM is a Turing machine in which every nonfinal state is either *universal* or *existential*. A configuration with an existential state is *accepting* if at least one successor configuration is

accepting. A configuration with a universal state is *accepting* if every successor configuration is accepting. An ATM accepts an input string if its initial configuration is accepting. An ATM has a two-way read-only input tape with endmarkers and k worktapes, which are initially blank. A step of an ATM consists of reading one symbol from each worktape and reading an input symbol, then writing a symbol on each of the worktapes, moving each of the heads left or right one tape square or not moving the tape heads, and choosing a new state from the set specified by the transition function.

One can describe all possible computations of an ATM on some input string as a *computation tree*. All nodes are configurations, the root is the initial configuration, and the children of any configuration c are exactly those configurations that can be reached from c in one step according to the transition rules of the ATM. The leaves of the tree are the final configurations and may be accepting or rejecting. A *branch* of the computation tree is a downward directed path from the root: in other words, a branch is a sequence of configurations starting with the initial configuration. Assume that for an ATM to run in time T(n), all branches terminate in at most T(n) steps.

Formally, an ATM AM is a septuple $AM = (k, Q, \Sigma, \Gamma, \delta, q_0, g)$ where

k is the number of worktapes,

Q is a finite set of states,

Σ is a finite input alphabet ($\$ \notin \Sigma$ is an endmarker),

Γ is a finite worktape alphabet ($B \in \Gamma$ is the blank symbol),

δ: $Q \times \Gamma^k \times (\Sigma \cup \{\$\}) \rightarrow P(Q \times (\Gamma - \{B\})^k \times \{left, right, stationary\}^{k+1})$ is the transition function, where P(S) is the power set of S, that is, the collection of subsets of S,

$q_0 \in Q$ is the initial state, and

g: $Q \rightarrow \{universal, existential, accept, reject\}$ is a mapping identifying each state as a universal, existential, accepting, or rejecting state.

Chapter 3

LITERATURE REVIEW

This chapter reviews the literature related to the research reported in this thesis.

Chandra, Kozen, and Stockmeyer (1981) present the concept of alternation. (The same authors originally presented the concept in Chandra and Stockmeyer (1976) and Kozen (1976).) This thesis uses their ATM model. They derive significant relationships between classes of languages accepted by time and space bounded ATMs and those accepted by time and space bounded DTMs. In particular, logarithmic alternating space is equivalent to polynomial deterministic time, and polynomial alternating time is equivalent to polynomial deterministic space. Paul, Prauss, and Reischuk (1980) demonstrate that an ATM with a single tape can simulate an ATM with multiple tapes in linear time.

Dymond and Tompa (1985) prove another result related to this thesis. They establish that DTM(t) $\subseteq$ ATM(t/log t). Their proof associates the computation of the DTM with an acyclic directed graph. They use a two-person pebbling game to pebble the graph within a time bound of O(n/log n) for a graph with n vertices. Next, the ATM steps simulate the two-person pebbling of the graph. In the pebbling game, one person's moves correspond to existential choices of the ATM, and the other person's moves correspond to universal choices of the ATM.

Paterson (1972) represents a TM computation as a two-dimensional diagram of successive tape configurations. He employs divide-and-conquer in both time and space dimensions. This method is generalized in this thesis in the simulation of a dIA by an ATM. Loui (1981) establishes a space bound for a DTM to accept the same language as a d-dimensional NTM with one worktape head. The proof utilizes a generalization of crossing sequences across the boundaries of d-dimensional boxes of the worktape. He uses a divide-and-conquer method to recursively partition the boxes.

Rosenfeld (1979) presents a good review of iterative automata.

Cole (1969) formally presents the d-dimensional iterative array of finite state machines. He establishes that the computing speed of a dIA can be increased by a constant factor by enlarging the set of states of each machine. He proves that the class of context-free languages does not contain all the sets of strings accepted by a dIA, nor do the sets of strings accepted by a dIA contain all context-free languages. He proves that computing capability increases as the number of dimensions increases.

Seiferas (1977a) extends Cole's work on deterministic dIAs to nondeterministic dIAs (NdIAs). He derives that

$$\mathrm{NTM}(t^d) \subseteq \mathrm{NdIA}(t),$$

$$\mathrm{NdIA}(t) \subseteq \mathrm{NTM}(t^{d+1}), \text{ and}$$

$$\mathrm{dIA}(t) \subseteq \mathrm{DTM}(t^{d+1}).$$

The second result is related to the simulation of an NdIA by an ATM given in this thesis. His simulation uses about n^d steps of a one-dimensional (2d+1) head TM to simulate the nth step in a computation of a dIA.

Seiferas (1977b) establishes that a dIA with direct central control is no more powerful than a regular dIA, and that a regular dIA can simulate a dIA with direct central control in linear time. In this simulation, the finite state machine at the origin of the dIA controls the dIA indirectly by propagating the value of its state outward using only local communication.

Culik and Yu (1984) construct a language L such that an ITA accepts L in real-time, but no dIA can accept L in real-time. They state that the converse problem, that is, whether real-time dIA languages ($d \geq 2$) are properly contained in real-time ITA languages, is an open problem. They establish that an ITA can simulate an NTM in linear time. Their simulation provides a basis for the simulation of an ATM by an ITA in Chapter 4.

Chapter 4

THE ITA SIMULATION OF THE DIA

This chapter contains a simulation of the ATM by the ITA and a simulation of the dIA by the ATM and proofs of the correctness of each simulation.

Lemma 1: Every language recognized in time t by a k-tape ATM can be recognized in time $O(t)$ by a one-tape ATM.

This result is from Paul, Prauss, and Reischuk (1980). They specify that the one-tape ATM does not have separate input and output tapes.

Lemma 2: Every t steps of an ATM with at most $c \geq 3$ choices at each step can be simulated by $(c-1)t$ steps of an ATM with at most 2 choices at each step.

Proof: Let $M = (k, Q, \Sigma, \Gamma, \delta, q_0, g)$ be an ATM with at most c choices at each step. We define an ATM $M' = (k, Q', \Sigma, \Gamma, \delta', q_0, g)$ with at most 2 choices at each step such that M' simulates M. Let a transition rule of M be

$$\delta(q_0, w_0, a_0) = \{(q_1, w_1, X_1), (q_2, w_2, X_2), \ldots, (q_j, w_j, X_j)\}$$

for $q_0, q_1, \ldots, q_j \in Q$, $w_0, w_1, \ldots, w_j \in \Gamma^k$, $a_0 \in \Sigma \cup \{\$\}$, $X_1, X_2, \ldots, X_j \in \{left, right, stationary\}$, $0 \leq j \leq c$. If $j \leq 2$, then $\delta'(q_0, w_0, a_0) = \delta(q_0, w_0, a_0)$. Otherwise, δ' has the following corresponding rules:

$$\delta'(q_0, w_0, a_0) = \{(q_1, w_1, X_1), (p_1, w_0, stationary)\}$$

$$\delta'(p_1, w_0, a_0) = \{(q_2, w_2, X_2), (p_2, w_0, stationary)\}$$

$$\vdots$$

$$\delta'(p_{j-2}, w_0, a_0) = \{(q_{j-1}, w_{j-1}, X_{j-1}), (q_j, w_j, X_j)\}.$$

The new state set Q' includes Q and all the new states $p_1, p_2, \ldots, p_{j-2}$ that are used in the

decomposition of the steps with three or more choices into steps with two choices.

It is clear that each move of M can be simulated by M' with at most $c-1$ moves. So $(c-1)t$ steps of M' are enough to simulate t steps of M. □

Definition: *Subtree Broadcast ITA* (SBITA) — An SBITA S is the same as an ITA except as specified in the following. Some of the machines in S are designated to be *control units*. If $S(\alpha)$ is a control unit, then for every descendant $S(\beta)$ of $S(\alpha)$, the next state of $S(\beta)$ depends on the current state of $S(\alpha)$ as well as on the current states of $S(\beta)$ and its parent and its children. At all times, for every machine $S(\beta)$, at most one ancestor of $S(\beta)$ is a control unit. During the computation a control unit $S(\alpha)$ ceases to function as a control unit when it designates its children $S(\alpha 0)$ and $S(\alpha 1)$ to become control units. Initially, $S(\lambda)$ is the only control unit.

Lemma 3: Every t steps of an SBITA can be simulated by $2t$ steps of an ITA.

Proof: Let S be an SBITA. We design an ITA R that simulates S.

$R(\beta)$ is designated as a control unit when the state of $R(\beta)$ is in a special set of states called the *broadcast set*. The broadcast set contains a particular state q_{pass}, and $R(\beta)$ enters state q_{pass} on the step before it will pass its control unit capabilities to its descendants. If $R(\beta)$ is in state q_{pass} at any time t, then at time t+1, $R(\beta)$ will be in some state not in the broadcast set. At any time t, the state of $R(\beta)$ can be in the broadcast set only if one of the following is true:

(1) $\beta=\lambda$ and t=0,

(2) at time t-1, the state of $R(\beta)$ was in the broadcast set, or

(3) at time t-1, the parent of $R(\beta)$ was in state q_{pass}.

The remainder of the proof is taken directly from Seiferas (1977b), and is modified to apply to deterministic ITAs rather than nondeterministic dIAs. Informally, a machine in R

simulating a control unit in S performs a broadcast by propagating the values of its states to its descendants. The simulation of each descendant will lag by time equal to its distance from the control unit, so communication between a control unit and any descendant will take twice as long in R as in S.

Let Q' denote the set of states of each machine in S. Let δ'_0, δ'_1, and δ'_2 denote the transition functions of S.

Let Q denote the set of states in R, where $q_{\lambda}=(\{0,1\}\times q_{\lambda}\times q_{\lambda}\times q_{\lambda})$. $Q=(\{0,1\}\times Q'\times Q'\times Q')\cup q_{\lambda}$. When $q\in Q$, denote the first component as *phase(q)*, the second component as *control(q)*, the third component as *prev(q)*, and the fourth component as *current(q)*. *Control* (q) propagates the states of the control unit to its descendants; *prev* (q) holds the previous state of a machine for reference by its children; *current* (q) holds the current state in the simulation of the corresponding machine in S.

Define for $R(\lambda)$ as control unit

$$\delta_0(q_x, a, q_l, q_r) = \begin{cases} (0, \delta'_0(control(q_x), current(q_x), a, q'_l, q'_r), current(q_x), \\ \quad \delta'_0(control(q_x), current(q_x), a, q'_l, q'_r)) \\ \quad \text{if } q_x \in Q - q_{\lambda}, phase(q_x) = 1. \\ (1, control(q_x), prev(q_x), current(q_x)) \\ \quad \text{if } q_x \in Q - q_{\lambda}, phase(q_x) = 0. \ r(q_x, a, q_l, q_r) = false. \\ (1, phase(q_x), phase(q_x), phase(q_x)) \\ \quad \text{if } q_x = q_{\lambda}. \ r(q_x, a, q_l, q_r) = false. \ or \\ \varnothing \\ \quad otherwise. \end{cases}$$

$$where \quad q'_i = \begin{cases} phase(q_i) & \text{if } q_i = q_{\lambda} \ or \\ current(q_i) & otherwise. \end{cases}$$

Define for $R(\beta)$, $\beta\neq\lambda$, where $R(\beta)$ is a control unit.

$$\delta_l(q_\lambda, q_l, q_r, q_p) = \begin{cases} (0, \delta'_l(control(q_\lambda).current(q_\lambda).q'_l.q'_r.q'_p), current(q_\lambda), \\ \quad \delta'_l(control(q_\lambda).current(q_\lambda).q'_l.q'_r.q'_p)) \\ \quad \text{if } q_\lambda \in Q - q_{\lambda\lambda}.phase(q_\lambda)=1. \\ (1.control(q_\lambda).prev(q_\lambda).current(q_\lambda)) \\ \quad \text{if } q_\lambda \in Q - q_{\lambda\lambda}.phase(q_\lambda)=0. \\ (1.phase(q_\lambda).phase(q_\lambda).phase(q_\lambda)) \\ \quad \text{if } q_\lambda = q_{\lambda\lambda}. or \\ \varnothing \\ \quad otherwise. \end{cases}$$

$$where \quad q'_i = \begin{cases} phase(q_i) & \text{if } q_i = q_{\lambda\lambda} or \\ current(q_i) & otherwise. \end{cases}$$

δ_r is defined similarly.

Define for $R(\beta).\beta \neq \lambda$, where $R(\beta)$ is not a control unit.

$$\delta_l(q_\lambda, q_l, q_r, q_p) = \begin{cases} (0.control(q_p).current(q_\lambda), \delta'_l(control(q_\lambda).current(q_\lambda).q'_l.q'_r.q'_p)) \\ \quad \text{if } q_\lambda.q_p \in Q - q_{\lambda\lambda}.phase(q_\lambda)=1. \\ \quad (Note: This\ transition\ propagates\ control\ information.) \\ (1.control(q_\lambda).prev(q_\lambda).current(q_\lambda)) \\ \quad \text{if } q_\lambda \in Q - q_{\lambda\lambda}.phase(q_\lambda)=0. \\ (1.control(q_p).phase(q_\lambda).phase(q_\lambda)) \\ \quad \text{if } q_\lambda = q_{\lambda\lambda}.q_p \in Q - q_{\lambda\lambda}. or \\ \varnothing \\ \quad otherwise.. \end{cases}$$

$$where \quad q'_i = \begin{cases} phase(q_i) & \text{if } q_i = q_{\lambda\lambda} \\ prev(q_i) & \text{if } q_i \in Q - q_{\lambda\lambda}.i = p. or \\ current(q_i) & otherwise. \end{cases}$$

δ_r is defined similarly.

Informally, the first three cases in each definition above are "simulate a transition," "wait your turn to simulate a transition," and "set up to start simulation," respectively.

This completes the definition of ITA R that simulates SBITA S.

A configuration of R is a pair (C,w); a configuration of S is a pair (C',w).

The sequence $\alpha=[(C_0,w_0),(C_1,w_1),(C_2,w_2),\ldots]$ of configurations of R is *skewed* if all of the following hold for all nonnegative integers j, $\beta \in \{0,1\}^*$:

$w_{2j}=w_{2j+1}$.

if $j \leq |\beta|$, then $C_j(\beta)=C_0(\beta)=q_1$.

if $j > |\beta|$, then $C_j(\beta)\in Q$.

$$phase(C_j(\beta)) = \begin{cases} 1 & \text{for } j-|\beta| \text{ odd}, \\ 0 & \text{for } j-|\beta| \text{ even}. \end{cases}$$

$control(C_{|\beta|+j+1}(\beta))=current(C_{j+1}(\lambda))$.

$$\begin{aligned} prev(C_{|\beta|+1}(\beta)) &= current(C_{|\beta|+1}(\beta)) \\ &= prev(C_{|\beta|+2}(\beta)) \\ &= prev(C_{|\beta|+3}(\beta)) \\ &= phase(C_0(\beta)), \text{ and} \end{aligned}$$

$$\begin{aligned} current(C_{|\beta|+2j+2}(\beta)) &= current(C_{|\beta|+2j+3}(\beta)) \\ &= prev(C_{|\beta|+2j+4}(\beta)) \\ &= prev(C_{|\beta|+2j+5}(\beta)). \end{aligned}$$

For such a skewed sequence α, define

$$skew^{-1}(\alpha)=[(C'_0,w'_0),(C'_1,w'_1),(C'_2,w'_2),\ldots]$$

if the following hold for all nonnegative integers j, $\beta \in \{0,1\}^*$, $q=C_{|\beta|+2j}(\beta)$:

$w'_j=w_{2j}$ and

$$C'_j(\beta)=\begin{cases} phase(q) & \text{if } q=q_1 \text{ or} \\ current(q) & otherwise. \end{cases}$$

The definition of a skewed sequence does not restrict the choice of values $current(C_{|\beta|+2j}(\beta))$ for $j>0$, so every sequence (C'_0,w'_0), (C'_1,w'_1), $(C'_2,w'_2),\ldots$ of configurations of S where all machines are initially quiescent is equal to $skew^{-1}(\alpha)$ for some skewed sequence α. We show below that α is a computation by R from (C_0,w_0) if and only if $skew^{-1}(\alpha)$ is a computation by S from (C_0,w_0). It will suffice to map state $q \in Q$ to state

phase $(q) \in Q'$ if $q = q_\Lambda$ or to state *current* $(q) \in Q'$ otherwise.

Let α be a skewed sequence with

$\alpha = [(C_0, w_0), (C_1, w_0), (C_2, w_1), (C_3, w_1), \ldots]$ and

$skew^{-1}(\alpha) = [(C'_0, w_0), (C'_1, w_1), (C'_2, w_2), \ldots]$.

To prove that α is a computation by R if and only if $skew^{-1}(\alpha)$ is a computation by S, it suffices to verify for all nonnegative integers j, $\beta \in \{0,1\}^* \cup \{\lambda\}$, $y = |\beta|$ or $y = |\beta| + 2j + 2$, that $C_{y+1}(\beta)$ results from the transition from LC(β, y) and that $C_{|\beta|+2j+2}(\beta)$ results from the transition from LC(β, $|\beta| + 2j + 1$) if and only if $C'_{j+1}(\beta)$ results from the transition from LC(β,j). □

Lemma 4: Every t steps of a one-head, one-tape TM can be simulated by a two-stack machine in up to $5t$ steps.

Proof: This lemma is proved in Hopcroft and Ullman (1979).

Call the stacks of the two-stack machine stack A and stack B. Let tp(1), ..., tp(j) denote the nonblank contents of the TM tape. The two-stack machine *represents* a TM configuration by holding tp(1), ..., tp(i) in stack A with tp(i) at the top and tp(i+1), ..., tp(j) in stack B with tp(i+1) at the top, where tp(i) is the tape element that the head of the TM is reading.

In five steps, the two-stack machine can simulate a TM step in which the head moves to the right:

1. Pop tp(i) from stack A.
2. Change the symbol contained in tp(i) to the symbol to be written in this step.
3. Push tp(i) back onto stack A.
4. Pop tp(i+1) from stack B.
5. Push tp(i+1) onto stack A.

In three steps, the two-stack machine can simulate a TM step in which the head moves to the left:

1. Pop tp(i) from stack A.
2. Change the symbol contained in tp(i) to the symbol to be written in this TM step.
3. Push tp(i) onto stack B.

In three steps, the two-stack machine can simulate a TM step in which the head remains on the same tape element:

1. Pop tp(i) from stack A.
2. Change the symbol contained in tp(i) to the symbol to be written in this TM step.
3. Push tp(i) back onto stack A. □

Theorem 5: For all $T(n)$, every language recognized in time $T(n)$ by a k-tape ATM can be recognized by an ITA in time $O(T(n))$.

Proof: Let AM be a one-tape ATM with at most two choices at each step. By Lemmas 1, 2, and 3, to prove the theorem it suffices to show that an SBITA S can simulate AM in time O(T(n)).

It is clear that a one-head, one-tape TM can directly implement a branch of the computation tree of AM. By Lemma 4, a two-stack machine can simulate the actions of a one-head, one-tape TM.

S can implement the stacks in the manner described below (modeled after the implementation in Culik and Yu (1984)). For all β and all $z>0$, all descendants of $S(\beta)$ at distance z from $S(\beta)$ are in the same state and contain the same stack elements. Let $\Delta(\beta)$ denote the portion of S consisting of $S(\beta)$ and all of its descendants. $\Delta(\beta)$ implements a pair of stacks, stack A and stack B, each of which operates separately according to the rules below. Every machine in $\Delta(\beta)$ is a finite state machine and can store one stack element of each stack. $S(\beta)$ is a control unit and holds the state of the two-stack machine and the

elements at the top of each stack. To *push* stack A, $S(\beta)$ broadcasts to each of its descendants $S(\alpha)$ to send the element of stack A contained in $S(\alpha)$ to both children of $S(\alpha)$, and $S(\beta)$ stores the element to be pushed. To *pop* stack A, $S(\beta)$ broadcasts to each of its descendants $S(\alpha)$ to send the element of stack contained in $S(\alpha)$ to the parent of $S(\alpha)$.

At the beginning of the simulation, by definition of an SBITA, $S(\lambda)$ is the only control unit. $\Delta(\lambda)$ is functioning as a pair of stacks, A and B, to store the input string w. $S(\lambda)$ reads w and stores w in stack A. $S(\lambda)$ next pops w from stack A and pushes w onto stack B. At this point, w is stored in stack B with the first symbol of w at the top of stack B. In addition, $\Delta(\lambda)$ is functioning as a pair of stacks, C and D, to simulate the actions in a branch of the computation tree of AM. This simulation continues as long as each step in the computation tree of AM has only one choice. When AM first takes a step with two choices, control unit $S(\lambda)$ broadcasts to its descendants to push all stacks one level down, and $S(\lambda)$ passes the control unit capability to $S(0)$ and $S(1)$. Then $\Delta(0)$ simulates one of the choices, and $\Delta(1)$ simulates the other choice. $S(\lambda)$ no longer acts as a control unit, but enters existential state q_e if AM is in an existential state at this point or enters universal state q_u if AM is in a universal state at this point. Whenever two choices are present at a node of the binary computation tree, S will proceed to simulate each of the choices in the manner described above except the steps will be for $\Delta(\beta)$ instead of for $\Delta(\lambda)$. S simulates each branch of the binary computation tree in parallel with the other branches.

When a branch finishes a computation, the unique control unit for that branch sends the answer (acceptance/rejection) up to its parent, $S(\beta)$, $\beta \in \{0,1\}^*$. If $S(\beta)$ is in state q_e, it will enter state q_{acc}, indicating acceptance, or enter state q_{rej}, indicating rejection, based on the table in Figure 4.1; if $S(\beta)$ is in state q_u, it will enter state q_{acc} or state q_{rej} based on the table in Figure 4.2.

To determine the time required for S to simulate AM, look at the various components of the simulation. The reading of input string w and positioning of w in stack B by $S(\lambda)$

state of $R(\beta 1)$ / state of $R(\beta 0)$	q_{acc}	q_{other}	q_{rej}
q_{acc}	q_{acc}	q_{acc}	q_{acc}
q_{other}	q_{acc}	q_e	q_e
q_{rej}	q_{acc}	q_e	q_{rej}

Figure 4.1 - Table of state transitions from state q_e.
q_{other} represents any state other
than q_{acc} or q_{rej}.

state of $R(\beta 1)$ / state of $R(\beta 0)$	q_{acc}	q_{other}	q_{rej}
q_{acc}	q_{acc}	q_u	q_{rej}
q_{other}	q_u	q_u	q_{rej}
q_{rej}	q_{rej}	q_{rej}	q_{rej}

Figure 4.2 - Table of state transitions from state q_u .

requires time 2n. By Lemma 4. the two-stack simulation of each branch requires time O(T(n)). The concurrent SBITA simulation of two-stack machines requires time T(n). The passing of the result up to the root requires time T(n). Therefore the overall simulation of AM by S requires time O(T(n)). □

Next is the ATM simulation of the dIA.

Theorem 6: For all $T(n)$, every language recognized in time $T(n)$ by a dIA can be recognized in time $O((T(n))^d)$ by an ATM.

Proof: Let M be a nondeterministic dIA of time complexity T(n). We design an ATM AM that will simulate the operation of M. (Note: AM will existentially guess T(n).)

The *local configuration* of M($\underline{X}$) at time t. denoted LC($\underline{X}$,t). is the (2d+1)-tuple containing the state of M($\underline{X}$) at time t and the states of the 2d neighbors of M($\underline{X}$) at time t.

The computation of M on the input can be expressed by a (d+1)-dimensional computation array C in which the value at coordinates ($\underline{X}$,t) is the state of M($\underline{X}$) at time t. except for one column of C that contains the input string. (See Figure 4.3.) For all d-dimensional subarrays D of C denote the endpoints of D as x_{1a} and x_{1b} in dimension 1. x_{2a} and x_{2b} in dimension 2. .... x_{da} and x_{db} in dimension d. and t_a and t_b in dimension d+1. where $x_{1a} < x_{1b}$, $x_{2a} < x_{2b}$, ... $t_a < t_b$. For k=1.....d. let X_k denote $\{x_{ka}-1, x_{ka}, \ldots, x_{kb}, x_{kb}+1\}$. Let X_k' denote $\{x_{ka}, \ldots, x_{kb}\}$. Let $(x_{1a}, X_2', X_3', \ldots, X_{d+1}')$ denote the set of coordinates

$$\{(x_{1a}, x_{2a}-1, x_{3a}-1, \ldots, t_a),$$

$$(x_{1a}, x_{2a}, x_{3a}-1, \ldots, t_a), \ldots,$$

$$(x_{1a}, x_{2b}+1, x_{3a}-1, \ldots, t_a),$$

$$(x_{1a}, x_{2a}-1, x_{3a}, \ldots, t_a), \ldots,$$

$$(x_{1a}, x_{2b}+1, x_{3a}, \ldots, t_a), \ldots,$$

$$(x_{1a}, x_{2b}+1, x_{3b}+1, \ldots, t_b)\}.$$

Let ∂D be the *boundary* of D. ∂D contains a value for each coordinate vector in the union of

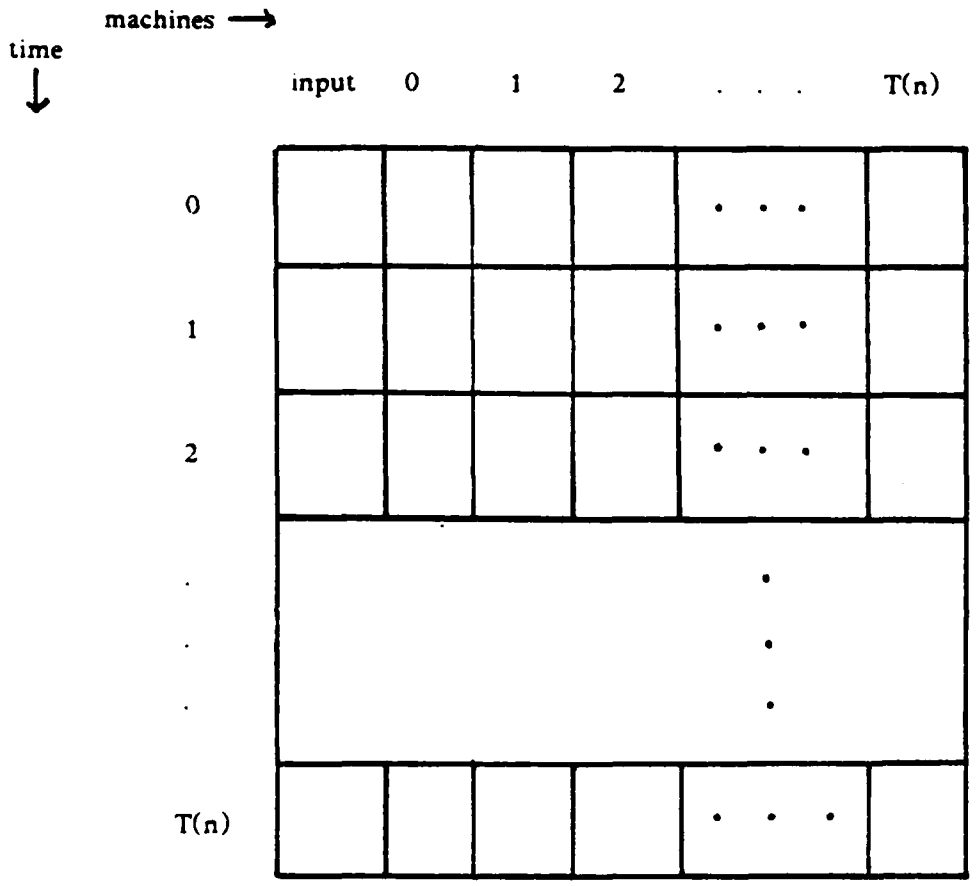


Figure 4.3 - Computation array C for 1-dimensional case

the sets

$(x_{1a}-1, X_2, \ldots, X_d, X_{d+1}')$,

$(x_{1a}, X_2, \ldots, X_d, X_{d+1}')$,

$(x_{1a}+1, X_2, \ldots, X_d, X_{d+1}')$,

$(x_{1b}-1, X_2, \ldots, X_d, X_{d+1}')$,

$(x_{1b}, X_2, \ldots, X_d, X_{d+1}')$,

$(x_{1b}+1, X_2, \ldots, X_d, X_{d+1}')$,

$(X_1, x_{2a}-1, X_3, \ldots, X_d, X_{d+1}')$, ...

$(X_1, X_2, \ldots, x_{db}+1, X_{d+1}')$,

$(X_1, \ldots, X_d, t_a)$, and

$(X_1, \ldots, X_d, t_b)$.

Informally, ∂D consists of $6d+2$ sets of values around the boundary of D. The values at the corners of D are contained in multiple sets. Each set of values is contained on a separate tape. ∂C contains the input string.

(Note: For d=1, ∂D is shown in Figure 4.4.)

Assume that when M($\underline{Q}$) enters an accepting state, then on the following steps M($\underline{Q}$) remains in the accepting state, and every other machine in M proceeds to enter the quiescent state.

Define *valid* (∂D) to be a predicate that is true if and only if there is an assignment of states to the entries of D such that all state transitions of each machine in M follow from the transition rules of M. Call such an assignment *consistent*. The recursive procedure ASIMD, defined below, computes the predicate *valid*, that is, ASIMD returns "*true*" or "*false*." ASIMD uses a divide-and-conquer method according to time and according to each dimension of the machines in M.

Initially, AM reads the input string, existentially inserts null symbols in the input string for the steps when M does not read an input symbol, existentially guesses the

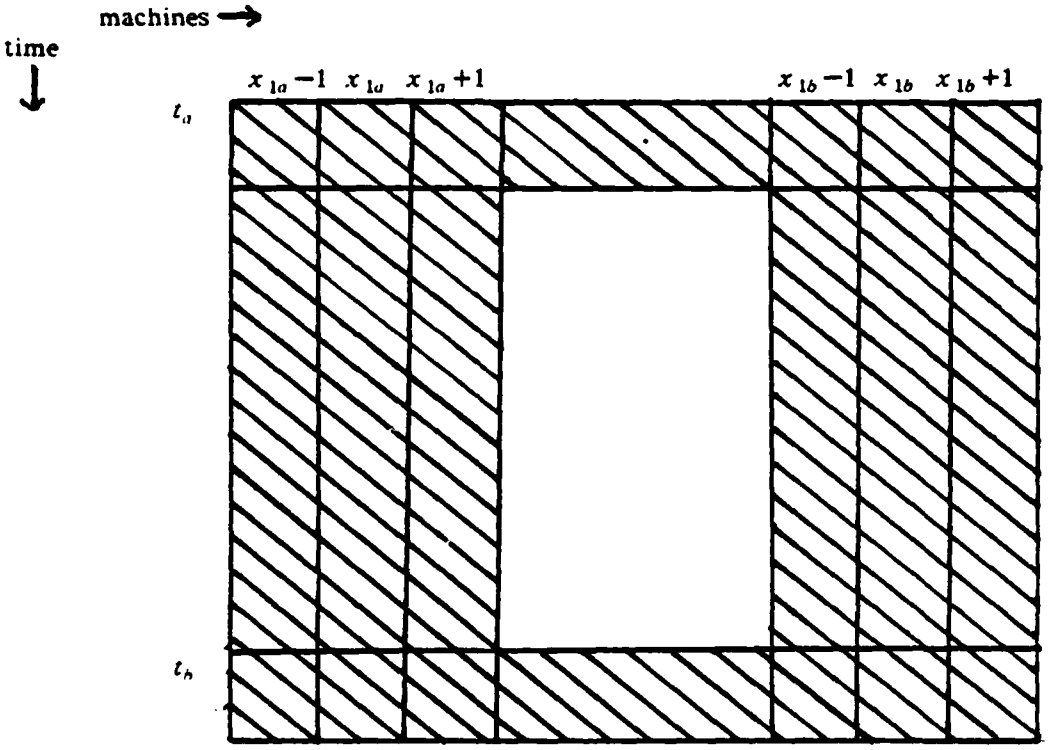


Figure 4.4 - ∂D , input to procedure ASIMD for 1-dimensional case

remainder of ∂C, and calls ASIMD(∂C). AM assumes that the length of every side of C is the smallest power of 2 greater than or equal to T(n). AM accepts the input string if and only if

(1) ASIMD(∂C) returns "true." and

(2) the state of M($\underline{0}$) at time T(n) determined by ASIMD(∂C) is an accepting state.

The recursive procedure ASIMD(∂D) is defined as follows:

Input: ∂D.

Input invariant: There exists an integer j such that for all i $x_{ib} - x_{ia} = 2^j$ and $t_b - t_a = 2^j$. Call 2^j the *width* of D.

Output: "true" or "false"

Step 1: If the width of D is 1, then check that for all $\underline{Y}$ representing coordinates in D, the state of M($\underline{Y}$) at time $t_b = t_a + 1$ correctly follows from LC($\underline{Y}$,t_a) according to the transition rules of M. If this condition holds, then the answer returned by this invocation of ASIMD is "true"; otherwise, the answer is "false."

Step 2: At this point the width of D is greater than 1. Let $t_m = (t_a + t_b)/2$. Existentially guess the states for all machines whose coordinates are in $(X_1^*, \ldots, X_d^*)$ at time t_m. For all machines whose state at time t_m is in ∂D, check that the state guessed in this step is equal to the state in ∂D. If the states are not equal, then return "false."

Step 3: For each i, $1 \leq i \leq d$, let $x_{im} = (x_{ia} + x_{ib})/2$. Existentially guess LC($\underline{Y}$,t) for all $\underline{Y} \in (X_1, X_2, \ldots, x_{im}, \ldots, X_d)$ and all $t_a \leq t \leq t_b$. Check that the states in $LC(\underline{Y}, t_a)$, $LC(\underline{Y}, t_b)$, and $LC(\underline{Y}, t_m)$ are equal to the corresponding states in ∂D and to the corresponding states guessed in Step 2. If any of the corresponding states are not equal, then return "false." (See Figure 4.5.)

Step 4: Universally choose one of the 2^{d+1} subarrays of D created from performing divide-and-conquer in each of the (d+1) dimensions of D.

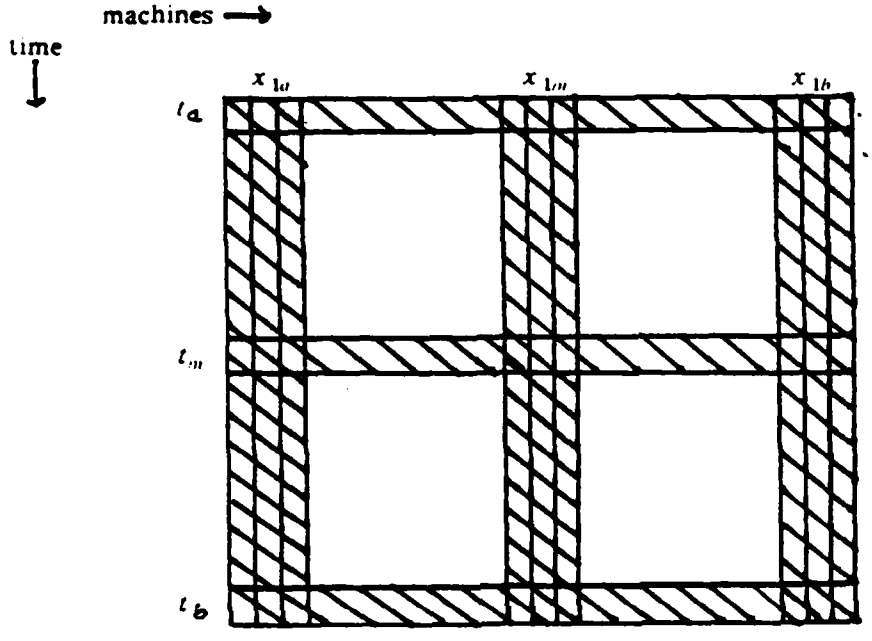


Figure 4.5 - Values known in computation array D after Step 3 of ASIMD

Step 5: Let S be the subarray chosen in Step 4. Call ASIMD(∂S).

Step 6: If all universal choices in Step 5 return "true," then return "true"; otherwise, return "false."

Now we prove that AM accepts if and only if M accepts.

First we show that $ASIMD(\partial C)$ returns "true" if and only if *valid* (∂C) is true. Let 2^j be the width of ∂D, the input to ASIMD. By induction on j we show that $ASIMD(\partial D)$ returns "true" if and only if *valid* (∂D) is true.

Basis (j=1): Step 1 of ASIMD confirms that each state at time t_b follows from the states at t_a according to the transition rules of M. Steps 2 and 3 confirm that state information on separate tapes for the same machine at the same time is equal. Therefore, *valid* (∂D) is true.

Conversely, if *valid* (∂D) is true, then the assignment of states to entries of $D \subseteq \partial D$ is consistent. Therefore, Step 1 of ASIMD confirms that each state at time t_b follows from the states at t_a according to the transition rules of M, and Steps 2 and 3 confirm that state information on separate tapes for the same machine at the same time is equal. Therefore, $ASIMD(\partial D)$ returns "true."

Inductive hypothesis: $ASIMD(\partial D)$ returns "true" if and only if *valid* (∂D) is true.

Induction (j>1): Assume $ASIMD(\partial D)$ returns "true." Let E, F, ... G be the subarrays of D from Step 4 of ASIMD. It follows that the calls to $ASIMD(\partial E)$, $ASIMD(\partial F)$, ... $ASIMD(\partial G)$ returned "true." By the inductive hypothesis, *valid* (∂E), *valid* (∂F), ... *valid* (∂G) are all true. Since array D is completely covered by the overlapping subarrays E, F, ... G, and the states assigned to overlapping locations of E, F, ... G are the same since they are passed to $ASIMD(\partial E)$, $ASIMD(\partial F)$, ... $ASIMD(\partial G)$ from $ASIMD(\partial D)$, and there is a consistent assignment of states to the entries of E, F, ... G, then there is a consistent assignment of states to the entries of D. Therefore, *valid* (∂D) is true.

Conversely, assume *valid* (∂D) is true. It follows that *valid* (∂E), *valid* (∂F), ..., *valid* (∂G) are all true. By the inductive hypothesis, *ASIMD* (∂E), *ASIMD* (∂F), ..., *ASIMD* (∂G) all return "true." Therefore, by Step 6 of ASIMD, *ASIMD* (∂D) returns "true."

Now we show that if AM accepts, then M accepts. If AM accepts input string w, then *ASIMD* (∂C) returns "true," and the state of M($\underline{0}$) at time T(n) determined by *ASIMD* (∂C) is an accepting state. Since *ASIMD* (∂C) returns "true," *valid* (∂C) is true. Because *valid* (∂C) is true and the state of M($\underline{0}$) at time T(n) determined by *ASIMD* (∂C) is an accepting state, an assignment of states to all machines in M for all t, $0 \leq t \leq T(n)$, that ends with M($\underline{0}$) in an accepting state exists, such that the states of all machines at time t result from the states of all machines at time $t-1$ according to the transition rules of M and input string w. This implies that M accepts w.

Next we show that if M accepts, then AM accepts. If M accepts input string w, then there exists an assignment of states to all machines in M for all t, $0 \leq t \leq T(n)$, such that the states of all machines in M at time t follow from the states of all machines at time $t-1$ according to the transition rules of M and the input string w, and such that all machines are initially quiescent, and such that M($\underline{0}$) is in an accepting state at time T(n). As a result, *valid* (∂C) is true; hence, *ASIMD* (∂C) returns "true." Because *ASIMD* (∂C) returns "true," and M($\underline{0}$) is in an accepting state at time T(n), AM accepts w.

Now we show that the time required for this simulation is $O((T(n))^d)$. Let D denote a (d+1)-dimensional subarray of length k in each dimension. Let $T_{AM}(k)$ denote the time complexity of the simulation. *ASIMD* (∂D) is performed on computation array D which has sides of length k. Then AM selects one of the subarrays S of D and calls *ASIMD* (∂S). The lengths of the sides of S are $k/2$, so *ASIMD* (∂S) requires time $T_{AM}(k/2)$. The time to perform Steps 2 and 3 of *ASIMD* (∂D) is $O(k^d)$. The time complexity of *ASIMD* (∂D) is

$$T_{AM}(k) = T_{AM}(k/2) + O(k^d) = O(k^d).$$

The space required for this simulation is $O(k^d)$. In particular, the simulation of M by AM

takes time $T_{AM}(T(n)) = O((T(n))^4)$ and space $O((T(n))^4)$. □

The same techniques can be used to simulate an NdIA on an ATM in the same time.

Corollary: For all $T(n)$, every language recognized in time $T(n)$ by an NdIA can be recognized in time $O((T(n))^4)$ by an ATM.

Chapter 5

THE ATM SIMULATION OF THE ITA

This chapter contains a simulation of the ITA by the ATM and a proof of the correctness of the simulation. It also outlines a simulation of an ATM by a dIA.

Theorem 7: For all $T(n)$, every language recognized in time $T(n)$ by an ITA can be recognized in time $O((T(n))^2)$ by an ATM.

Proof: Let R be an ITA of time complexity T(n). We design an ATM AM with four worktapes that will simulate the operation of R. (Note: AM will existentially guess T(n).)

For $\beta \in \{0,1\}^*$ and $X \in \{0,1\}$, the *local configuration* of $R(\beta X)$ at time t, denoted $LC(\beta X, t)$, is the quadruple $(q_s(t), q_l(t), q_r(t), q_p(t))$ where

$q_s(t)$ is the state of $R(\beta X)$ at time t.

$q_l(t)$ is the state of $R(\beta X 0)$ at time t.

$q_r(t)$ is the state of $R(\beta X 1)$ at time t.

$q_p(t)$ is the state of $R(\beta)$ at time t if $\beta X \neq \lambda$ or $q_p(t)$ is the input symbol at time t if $\beta X = \lambda$.

Define $|\lambda| = 0$.

Let $\Delta(\beta)$ denote the portion of R that comprises $R(\beta)$ and all descendants of $R(\beta)$. Let σ_β denote the sequence of states $q_s(t)$ of $R(\beta)$ for $0 \leq t \leq T(n)$. Define *valid* $(\sigma_{\beta X}, \sigma_\beta, \beta X)$ to be a predicate that is true if and only if there is an assignment of states to $\Delta(\beta X)$ for every time t, $0 \leq t \leq T(n)$, such that for every i, $0 \leq i \leq T(n)$, the states of every machine in $\Delta(\beta X)$ at time i follows from the states of every machine in $\Delta(\beta X)$ at time $i-1$ according to the transition rules of R, $\sigma_{\beta X}$, and σ_β. Call such an assignment of states *consistent*. The recursive procedure ASIMT, defined below, computes the predicate *valid*, that is, ASIMT returns "*true*" or "*false*."

Initially, AM writes input string w onto tape 4, existentially inserting null symbols for the steps when M does not read an input symbol. AM then existentially guesses the states for $R(\lambda)$ at each time step from 0 to T(n), then calls $ASIMT(\sigma_\lambda, w, \lambda)$. AM accepts if and only if

(1) the output of $ASIMT(\sigma_\lambda, w, \lambda)$ is "true," and

(2) the state assigned to $R(\lambda)$ at time T(n) by $ASIMT(\sigma_\lambda, w, \lambda)$ is an accepting state.

The recursive procedure $ASIMT(\sigma_{\beta X}, \sigma_\beta, \beta X)$ is defined below.

Inputs: $\sigma_{\beta X}$ (the sequence of states $q_s(t)$) on tape 1, σ_β (the sequence of states $q_p(t)$) on tape 4, and βX.

Output: "true" or "false"

Step 1: If $|\beta X| = T(n)$, then return "true" if all states of $q_s(t)$ are quiescent; if not all states are quiescent, then return "false." If $|\beta X| < T(n)$, then perform the following. Existentially guess $\sigma_{\beta X 0}$ on tape 2. Existentially guess $\sigma_{\beta X 1}$ on tape 3. (See Figure 5.1.) Verify for each t, $0 \leq t \leq T(n)$, that $q_s(t)$ follows from $LC(\beta, t-1)$, defined on tapes 1-4, according to the transition rules of R. If any $q_s(t)$ does not follow, then return "false." Verify that the symbols generated for $q_p(t)$ are equal to the symbols already on tape 4. If they are not, then return "false." If no consistent sequence of guesses is possible, then return "false."

Step 2: Copy the contents of tape 1 onto tape 4.

Step 3: Universally choose either $R(\beta X 0)$ or $R(\beta X 1)$. If $R(\beta X 0)$ is chosen, then copy the contents of tape 2 onto tape 1, then call $ASIMT(\sigma_{\beta X 0}, \sigma_{\beta X}, \beta X 0)$. If $R(\beta X 1)$ is chosen, then copy the contents of tape 3 onto tape 1, then call $ASIMT(\sigma_{\beta X 1}, \sigma_{\beta X}, \beta X 1)$.

Step 4: If both universal choices in Step 3 return "true," then return "true"; otherwise return "false."

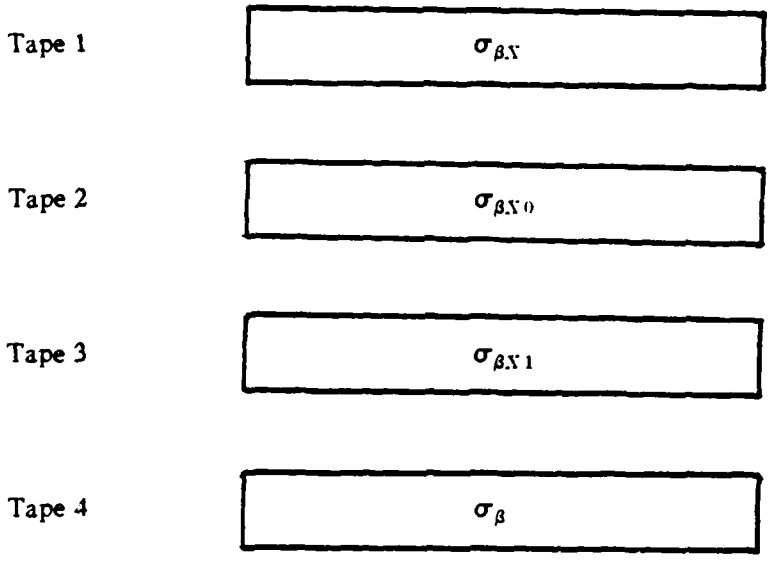


Figure 5.1 - Contents of ATM tapes in procedure ASIMT

Now it will be shown that AM accepts if and only if R accepts.

We show by induction on $h = T(n) - |\beta X|$, that $ASIMT(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ returns "true" if and only if $valid(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ is true.

Basis (h=0): If $ASIMT(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$, $|\beta X| = T(n)$, returns "true," then the sequence of local configurations is consistent with all descendants of $R(\beta X)$ according to the transition rules of R, since all machines in $\Delta(\beta X)$ are in a quiescent state at all times and have no effect on the computation. This implies that $valid(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ is true.

Conversely, if $valid(\sigma_{\beta_1}, \sigma_{\beta}, \beta X)$ is true, then a consistent assignment of states to $\Delta(\beta X)$ exists. According to the transition rules of R, all states assigned to $\Delta(\beta X)$ must be quiescent, because all machines in $\Delta(\beta X)$ are below level T(n) in R. Therefore, Step 1 of $ASIMT(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ confirms that all states in $\sigma_{\beta X}$ are quiescent. Therefore, $ASIMT(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ returns "true."

Inductive hypothesis: $ASIMT(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ returns "true" if and only if $valid(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ is true.

Induction (h>0): Assume $ASIMT(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ returns "true." It follows that for some $\sigma_{\beta X 0}$ and $\sigma_{\beta X 1}$ the calls to $ASIMT(\sigma_{\beta X 0}, \sigma_{\beta X}, \beta X 0)$ and $ASIMT(\sigma_{\beta X 1}, \sigma_{\beta X}, \beta X 1)$ returned "true." By the inductive hypothesis, $valid(\sigma_{\beta X 0}, \sigma_{\beta X}, \beta X 0)$ and $valid(\sigma_{\beta X 1}, \sigma_{\beta X}, \beta X 1)$ are both true. By Step 1 of $ASIMT(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$, $\sigma_{\beta X}$ is a sequence of states $q_1(t)$ for $R(\beta X)$, such that $q_1(t)$ follows from $LC(\beta X, t-1)$ according to the transition rules of R. Therefore, a consistent assignment of states to $\Delta(\beta X)$ exists. Therefore, $valid(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ is true.

Conversely, assume $valid(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ is true. It follows that both $valid(\sigma_{\beta X 0}, \sigma_{\beta X}, \beta X 0)$ and $valid(\sigma_{\beta X 1}, \sigma_{\beta X}, \beta X 1)$ are true. By the inductive hypothesis, both $ASIMT(\sigma_{\beta X 0}, \sigma_{\beta X}, \beta X 0)$ and $ASIMT(\sigma_{\beta X 1}, \sigma_{\beta X}, \beta X 1)$ return "true." Therefore, by Step 4 of ASIMT, $ASIMT(\sigma_{\beta X}, \sigma_{\beta}, \beta X)$ returns "true."

Now we show that if AM accepts, then R accepts. If AM accepts input string w, then $ASIMT(\sigma_{\lambda}, w, \lambda)$ has returned "true," and the state assigned to $R(\lambda)$ at time T(n) by

$ASIMT(\sigma_\lambda, w, \lambda)$ is an accepting state. Because $ASIMT(\sigma_\lambda, w, \lambda)$ returns "true," $valid(\sigma_\lambda, w, \lambda)$ is true. Since $valid(\sigma_\lambda, w, \lambda)$ is true, and since the state assigned to $R(\lambda)$ at time T(n) is an accepting state, R accepts w.

Next we show that if R accepts, then AM accepts. If R accepts input string w, then there exists an assignment of states to all machines in R for all t, $0 \leq t \leq T(n)$, such that the states of all machines in R at time t follow from the states of all machines at time $t-1$ according to the transition rules of R and input string w, and such that all machines are initially quiescent, and such that $R(\lambda)$ is in an accepting state at time T(n). As a result, $valid(\sigma_\lambda, w, \lambda)$ is true. Since $valid(\sigma_\lambda, w, \lambda)$ is true, $ASIMT(\sigma_\lambda, w, \lambda)$ returns "true." Because $ASIMT(\sigma_\lambda, w, \lambda)$ returns "true," and the state of $R(\lambda)$ at time T(n) is an accepting state, AM accepts w.

Now we show that the time required for this simulation is $O((T(n))^2)$. Let $T_{AM}(h)$ denote the time complexity of $ASIMT(\sigma_{\beta X}, \sigma_\beta, \beta X)$, where $h = T(n) - |\beta X|$. Steps 2 and 3 of $ASIMT(\sigma_{\beta X}, \sigma_\beta, \beta X)$ require time $O(T(n))$. $ASIMT(\sigma_{\beta X}, \sigma_\beta, \beta X)$ calls $ASIMT(\sigma_{\beta XY}, \sigma_{\beta X}, \beta XY)$, $Y \in \{0,1\}$, which has a time complexity of $T_{AM}(h-1)$. $T(0) = O(T(n))$. These terms give rise to the recurrence

$$T_{AM}(h) = T_{AM}(h-1) + O(T(n)).$$

In particular, the simulation of T(n) steps of R by AM requires time

$$T_{AM}(T(n)) = T_{AM}(T(n)-1) + O(T(n)) = O((T(n))^2).$$

This simulation requires space $O(T(n))$. □

In order to complete the cycle of simulations from ITA to dIA and from dIA to ITA, only the simulation of an ATM by a dIA remains. In order to obtain a time bound on this simulation, first consider a simulation of an NTM by a dIA. Suppose that a (deterministic) dIA operating in time $O(t_1^p)$, where p is some constant, could simulate t_1 steps of an NTM. Seiferas (1977a) proves that a DTM operating in time $O(t_2^{p+1})$ can simulate t_2 steps of a dIA. Together, these two simulations would imply that a DTM could simulate an NTM in

polynomial time; hence, the computational complexity class P would equal the computational complexity class NP. The equality P = NP is widely believed to be unlikely; hence,a polynomial time simulation of an NTM by a dIA is unlikely. Therefore, a polynomial time simulation of an ATM by a dIA is unlikely.

Proposition 8: For all T(n), every language recognized in time T(n) by an ATM can be recognized in time $O(2^{T(n)})$ by a dIA.

Briefly, a (deterministic) dIA can simulate an ATM in exponential time as follows. The dIA can simply compute each branch of the computation tree of the ATM in turn. Each branch corresponds to the actions of a DTM, and a dIA is able to simulate a DTM in linear time according to Seiferas (1977b).

Theorem 7 and Proposition 8 together yield an exponential time bound for a dIA simulation of an ITA. One would expect this bound because the number of finite state machines potentially involved in a computation grows polynomially with time for a dIA, but grows exponentially with time for an ITA.

Chapter 6

CONCLUSIONS AND OPEN PROBLEMS

This thesis has presented three simulations and discussed a fourth. When combined, Theorems 5 and 6 imply that an ITA can simulate a dIA in time $O(t^4)$, and Theorem 7 and Proposition 8 imply that a dIA can simulate an ITA in exponential time.

These results and the work done in obtaining them suggest several open problems.

1. Can the time bounds of Theorems 5, 6, and 7 be improved?

2. Is there a language L such that some dIA recognizes L in linear time, but every ITA requires superlinear time to recognize L? Culik and Yu (1984) pose this question, but for real-time. One candidate considered for L is a string of the form

$$L = (x_1 \# x_2 \# \ldots \# x_m \# \# y_1 \# y_2 \# \ldots \# y_m),$$

where $x_1, x_2, \ldots, x_m$ is an unordered list of values, and $y_1, y_2, \ldots, y_m$ is a sorted list of the same values. This candidate fails because an ITA can sort in time O(n) according to Browning (1979), and, though a 2IA can sort in time $O(\sqrt{n} \log n)$ according to Thompson and Kung (1977), Nassimi and Sahni (1979), and Stout (1982), the dIA must write down the output, leading to a total time requirement of O(n). A second possible candidate is a language of binary strings that represent connected d-dimensional figures.

3. How much time is required for an X-tree array to simulate an ATM? An X-tree is a binary tree with additional edges connecting all nodes at the same level in the tree. An X-tree array is an iterative array of finite state machines organized into an X-tree.

4. How much time is required for an ATM to simulate an X-tree array?

5. How can an ITA with depth as a function of the length of the input string simulate an ATM or an NTM?

6. How much time and space are required for an ATM with a limit on the number of its alternations to simulate either the dIA or the ITA?

A further possible area for future research is the alternating iterative array, such as either an alternating dIA or an alternating ITA. The addition of universal choices to non-deterministic dIAs or ITAs adds a second kind of parallelism. Initial work could be done in relating such a model to other models of computation.

REFERENCES

S. A. Browning (1979), "Computations on a Tree of Processors," *Caltech Conference on VLSI*, pp. 453-478, January 1979.

A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer (1981), "Alternation," *J. of the Association for Computing Machinery*, vol. 28, no. 1, pp. 114-133, January 1981.

A. K. Chandra and L. J. Stockmeyer (1976), "Alternation," *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*, pp. 98-108, October 1976.

S. N. Cole (1969), "Real-Time Computation by n-Dimensional Iterative Arrays of Finite-State Machines," *IEEE Trans. Comput.*, vol. C-18, no. 4, pp. 349-365, April 1969.

K. Culik II and S. Yu (1984), "Iterative Tree Automata," *Theoretical Computer Science*, vol. 32, no. 3, pp. 227-247, August 1984.

P. W. Dymond and S. A. Cook (1980), "Hardware Complexity and Parallel Computation," *Proceedings of the 21st IEEE Symposium on Foundations of Computer Science*, pp. 360-372, October 1980.

P. W. Dymond and M. Tompa (1985), "Speedups of Deterministic Machines by Synchronous Parallel Machines," *J. of Computer and System Sciences*, vol. 30, no. 2, pp. 149-161, April 1985.

S. Fortune and J. Wyllie (1978), "Parallelism in Random Access Machines," *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pp. 114-118, May 1978.

J. E. Hopcroft and J. D. Ullman (1979), *Introduction to Automata Theory, Languages, and Computation*, Reading, MA: Addison-wesley, 1979.

D. Kozen (1976), "On Parallelism in Turing Machines," *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*, pp. 89-97, October 1976.

M. C. Loui (1981), "A Space Bound for One-Tape Multidimensional Turing Machines (Note)," *Theoretical Computer Science*, vol. 15, no. 3, pp. 311-320, September 1981.

D. Nassimi and S. Sahni (1979), "Bitonic Sort on a Mesh-Connected Parallel Computer," *IEEE Trans. Comput.*, vol. C-27, no. 1, January 1979.

M. S. Paterson (1972), "Tape Bounds for Time-Bounded Turing Machines," *J. of Computer and System Sciences*, vol. 6, no. 2, pp. 116-124, April 1972.

W. J. Paul, W. J. Prauss, and R. Reischuk (1980), "On Alternation," *Acta Informatica*, vol. 14, no. 3, pp. 243-255, September 1980.

A. Rosenfeld (1979), *Picture Languages*, New York, NY: Academic Press, 1979.

W. J. Savitch (1970), "Relationships Between Nondeterministic and Deterministic Tape Complexities," *J. of Computer and System Sciences*, vol. 4, no. 2, pp. 177-192, April 1970.

J. I. Seiferas (1977a), "Linear-Time Computation by Nondeterministic Multidimensional

Iterative Arrays." *SIAM J. Comput.*, vol. 6, no. 3, pp. 487-504, September 1977.

J. I. Seiferas (1977b), "Iterative Arrays with Direct Central Control," *Acta Informatica*, vol. 8, no. 2, pp. 177-192, 1977.

Q. F. Stout (1982), "Using Clerks in Parallel Processing," *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pp. 272-279, November 1982.

C. D. Thompson and H. T. Kung (1977), "Sorting on a Mesh-Connected Parallel Computer," *Communications of the ACM*, vol. 20, no. 4, pp. 263-271, April 1977.

END

FILMED

1-86

DTIC